CINÉPHORIA

Documentation Technique

1. Architecture Globale

J'ai choisi de construire l'application Cinéphoria autour du framework Django, qui est écrit en Python. Ce framework m'a permis de structurer facilement l'application grâce à son système de modèles, vues, et templates, et surtout de bénéficier d'un ORM intégré très pratique.

L'application propose également une interface bureautique développée avec Electron.js, destinée aux employés pour signaler des incidents techniques. Django s'intègre facilement avec PostgreSQL pour les données relationnelles, Cloudinary pour les images (affiches de films, QR codes), et Heroku pour le déploiement. Ces choix techniques ont été motivés par leur accessibilité, leur documentation, et leur compatibilité avec un environnement de développement Dockerisé.

2. Architecture technique

Le fonctionnement général est organisé de manière simple : les utilisateurs interagissent avec le site web via leur navigateur. Lorsqu'ils effectuent une action, comme réserver une place ou se connecter, cela déclenche une requête qui est traitée par Django sur le serveur.

Le backend va ensuite communiquer avec différents services selon les besoins :

- Il utilise **PostgreSQL** pour stocker les informations importantes comme les utilisateurs, les films, les séances, etc.
- Il enregistre certaines données de statistiques dans **MongoDB**, comme le nombre de réservations par film. Ces données sont utilisées dans un tableau de bord destiné aux administrateurs.
- Il envoie les images (comme les affiches ou les QR codes) vers **Cloudinary**, un service de stockage en ligne spécialisé pour les médias.

Grâce à cette organisation, le projet est suffisamment souple pour évoluer, tout en étant clair et facile à maintenir.

3. Modèle de Données (BDD)

Le modèle de données est structuré autour de plusieurs entités principales : les films, les séances, les utilisateurs, les réservations, les billets et les sièges. Chaque billet est

lié à une seule réservation grâce à une relation OneToOne. À l'inverse, un film peut être projeté dans plusieurs cinémas, ce qui implique une relation ManyToMany.

Ces relations sont modélisées dans les fichiers du projet Django, et j'ai aussi produit un MCD (Modèle Conceptuel de Données), un MLD (Modèle Logique), que vous pouvez retrouver dans le dossier /docs du dépot. Ils m'ont aidée à mieux comprendre les dépendances entre les entités.

4. Fonctionnalités Clés

L'application permet à un visiteur de consulter les films à l'affiche et leurs séances. Une fois connecté, l'utilisateur peut réserver une séance, choisir ses sièges, et obtenir un billet avec un QR code. Après la séance, il peut laisser un avis.

Les employés disposent d'un intranet simplifié pour gérer les films, les séances et valider les avis. Une application bureautique a aussi été développée avec Electron pour leur permettre de signaler facilement les incidents dans les salles de cinéma (comme un siège cassé par exemple).

L'administrateur, quant à lui, a accès à un tableau de bord qui affiche des statistiques générales sur les réservations. Ces statistiques sont alimentées par une base de données MongoDB.

5. Tests et CI/CD

Pour m'assurer que les fonctionnalités de base fonctionnent correctement, j'ai écrit quelques tests automatiques avec le module de tests intégré à Django. Ils vérifient par exemple qu'une réservation peut bien être créée et qu'un billet est généré.

J'ai aussi configuré un fichier django.ylm dans .github/workflows pour que les tests soient lancés automatiquement à chaque mise à jour sur GitHub. Cela m'a permis de gagner du temps en détectant rapidement les erreurs.

Pendant le développement, j'ai utilisé Docker pour exécuter localement l'application ainsi que PostgreSQL, MongoDB et Redis. Le fichier docker-compose.yml permet de tout lancer en une seule commande. En production, j'ai utilisé Heroku qui facilite le déploiement.

Les tests permettent un taux de couverture de 74%.

6. Configuration Heroku / Environnement

Un fichier .env est indispensable pour indiquer les variables d'environnement nécessaires au fonctionnement de l'application. Sur Heroku, ces variables doivent être saisies dans la section Config Vars.

Voici un exemple de configuration minimale :

ALLOWED_HOSTS=lenomdel'app.herokuapp.com
SECRET_KEY=yourkeydjango
DEBUG=False
DJANGO_ENV=prod
DJANGO_SETTINGS_MODULE=cinephoria_app.settings.prod
DATABASE_URL=... (généré automatiquement par Heroku)
CLOUDINARY_URL=cloudinary://api:secret@cloudname
ALLOWED_HOSTS=*.herokuapp.com
MONGO_URI=mongodb+…

Chaque variable joue un rôle important : la clé secrète est utilisée par Django, Cloudinary sert à stocker les images, et DEBUG doit être désactivé en production pour des raisons de sécurité.

7. Transaction SQL

Pour valider mes compétences SQL, j'ai rédigé une transaction complète qui simule une réservation dans le système. Elle crée une réservation, lie deux sièges à celle-ci, puis génère un billet associé. Le tout est regroupé dans une transaction SQL avec BEGIN et COMMIT, pour garantir que si une étape échoue, rien n'est enregistré.

Cette transaction se trouve dans le fichier transaction_reservation.sql dans le dossier script du dépôt.

8. Gestion de Projet

Pour organiser mon travail, j'ai utilisé un tableau Kanban dans Trello, avec différentes colonnes pour les tâches à faire, en cours, et terminées. J'ai travaillé en suivant une approche agile avec des petits cycles de développement, ce qui m'a permis de livrer des fonctionnalités progressivement.

Pour le versionnage, j'ai utilisé Git avec des branches dédiées : une branche principale : « main », une branche de développement : « develop », et des branches pour chaque fonctionnalité créée à partir de la branche dévelop.

9. Environnement de travail

J'ai utilisé Visual Studio Code comme éditeur principal, avec quelques extensions utiles pour Python, Django, le formatage du code (Prettier) et la gestion de versions Git. Pour tester les routes API, j'ai utilisé Postman.

Pour gérer la base de données PostgreSQL, j'ai utilisé PgAdmin. Pour MongoDB, j'ai utilisé MongoDB Compass. Docker Desktop m'a permis de tout exécuter localement

sans rien installer de compliqué. Enfin, j’ai utilisé Firefox Developer Edition pour tester l’interface utilisateur dans le navigateur.

10. Livrables

Tous les livrables sont organisés dans le dépôt GitHub. Ils incluent :

- Un README.md expliquant comment installer, lancer et comprendre le projet
- Un dossier docs avec :
  - Une charte graphique
  - Un manuel utilisateur
  - Les modèles de données (MCD, MLD)
  - Un schémas de cas d’usages.
- Un script SQL de transaction
- Un script SQL de création de base
- Des captures d’écran et exports Figma pour montrer l’interface utilisateur

---

Projet réalisé dans le cadre de l’ECF 2025 - Concepteur Développeur d'Applications